## When Hive is best suited and when is it not?

Hive is best suitable in below scenarios:

1. For data warehouse application where relatively static data is analyzed, and data is not changing rapidly
2. Large data sets are maintained and mined for insights, reports.
3. Long running batch queries where reliability is critical
4. When the application on high latency

Hive is not suitable in below scenarios:

1. Data changes very frequently
2. Fast response time is required
3. Data volume is very less
4. Low latency queries
5. Start up is critical
6. If application require OLTP, switch to NoSQL database.

## When should one use Hive over MapReduce?

When the data is structured data then Hive should be use over MapReduce. In Hive, there are partitions to simplify data processing, bucketing for sampling & sorting the data quickly and these simplify the mapreduce process a lot. Also, partitions and buckets segment large datasets to improve query performance in hive.

## What is Hive metastore?

The Hive metastore service stores the metadata for Hive tables, and partitions in a relational database and provides clients (including Hive) access to this information via the metastore service API. Hive metastore contains information like IDs of Database, IDs of Tables, IDs of Index, time creation of an index, time creation of a table, IDs of roles assigned to a particular user, InputFormat used for a Table, OutputFormat used for a Table etc.

The Metastore will not know where in HDFS the data is present. It will use the IDs of tables or index and search in the namenode and the namenode will tell which block has the data.

The metastore service should run in an isolated JVM. Metadata (DDL = create/drop/rename tables, etc)) is stored in the metastore; the data is stored in the HDFS. This DDL-type metadata is different from the metadata that namenode stores. Hive metastore supports three execution backends: Spark, Tez and MapReduce.

## How can Hive improve performance with orc file format tables?

The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. ORC stores collections of rows in one file and within the collection the row data is stored in a columnar format. This allows parallel processing of row collections across a cluster. Each file with the columnar layout is optimized for compression and skipping of data/columns to reduce read and decompression load. ORC uses specific encoding like run-length encoding, dictionary encoding for strings and bitmap encoding for different column data types to improve compression. ORC file improves performance when Hive is reading, writing, and processing data. It reduces the data storage format by up to 75% of the original. ORC takes less time to access data and takes less space to store data. orc.compress indicates the compression techniques like Snappy,LZO etc, orc.stripe.size indicates blocks size of file, orc.row.index.stride indicates index. A larger block size of 256 MB by default optimizes for large sequential reads on HDFS for more throughput and fewer files to reduce load on the namenode. Hive 12 and later has execution layers to accelerate queries, both from the point of view of dealing with larger datasets and lower latencies.

**Advantages With ORC File Format**

i) Column stored separately

ii) Stores statistics (Min,Max,Sum,Count)

iii) Has Light weight Index

iv) Larger Blocks by default 256 MB

v) Reduce the accessing Time and storage Space

## What is thrift server and client, jdbc and odbc driver importance in hive?

**Thrift**: Thrift is a cross language Remote Procedure Call framework which generates code and combines a software stack and finally execute the Thrift code in a remote server. Thrift compiler acts as interpreter between server and client. Thrift server allows a client to submit request to Hive. The Hive server is exposed as a Thrift service to thrift server, so it is possible to interact with Hive using any programming language that supports Thrift like Python, Ruby, and Scala.

**JDBC Driver**: JDBC drive is a software component that enables a Java application to interact with a database. Using JDBC driver, java application connects to a Hive server running in a separate process at given host and port.

**ODBC driver:** ODBC driver acts as a translation layer between the applications and the DBMS. An ODBC driver allows applications that support ODBC protocol to connect to hive such as BI software.

## What is the importance of partition in hive?

When we query in Hive, Hive reads the entire data sets. It takes long time and heavy bottleneck for MapReduce jobs. To overcome this issue, Hive allows a special option called Partition the table. Based on the values of particular columns, hive segregates the input records into different files. Instead of analyzing vast amount of data, we can partition and analyze the target data to get desired results. And it is the best approach to improve query performance on larger tables. For example we have claims for whole year, whereas to process only month May claims, simply partition by the month. So instead of processing all claims, we have to process fewer amounts of data and get quick results.

Note: Do not partition too many columns, which is an overhead to NameNode.

There are two types of partition in Hive, static partition and Dynamic partition.

To do Dynamic partition set below two properties:

SET hive.exec.dynamic.partition = true;

SET hive.exec.dynamic.partition.mode = nonstrict;

Please note: Do not include partition columns in the table definition.

You need to do at least one static partition in the table for better performance.

The main difference between static and dynamic partition is hive.mapred.mode = strict or nonstrict

To debug, use DESCRIBE EXTENDED FORMATTED partition\_table\_name.

You need to load first temporary un-partitioned table to load partiioned table through HQL: INSERT OVERWRITE TABLE table\_name PARTITION (partition\_column1, partiion\_column2) select \* from un-partitioned\_table.

## What is the use of bucketing in hive?

Partitioning in Hive offers a way of segregating hive table data into multiple files/directories. But partiioning give effective results when,

* There are limited number of partitions
* Comparatively equal sized partitions

But this is not always possible.

To overcome the problem of over partitioning, Hive provides Bucketing concept. Bucketing is another technique for decomposing table data sets into more manageable parts.

Features:

* Bucketing concept is based on (hashing function on the bucketed column) mod(by total number of buckets). The hash function depends on the type of bucketing column.
* Records with same bucketed column will always be shared in the same bucket
* We use CLUSTERED BY clause to divide the table into buckets
* Physically, each bucket is just a file in the table directory
* Bucketing can be done along with partitioning and even without partitioning
* Bucketed tables creates almost equally distributed data file parts

Advantages:

* Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.
* As the data files are equal sized parts, map side join will be faster on bucketed tables than non-bucketed tables. In Map side join, a mapper processing a bucket of the left table knows that the matching rows in the right table will be on its corresponding bucket, so it only retrieves that bucket which is a small fraction of all the data stored in the right table
* Bucketing table provides faster query responses than non-bucketed tables.
* Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map side join even more efficient.

Note: Bucketed columns are included in table definition.

Note: We cannot load partitioned table directly.

SET hive.enforce.bucketing=true; NOTE: It will automatically set the number of reduce tasks to be equal to the number of buckets. Otherwise manually set mapred.reduce.tasks=32 and cluster by(state) SORT BY(city)

## What is the difference between static partitioning and dynamic partitioning in hive?

**Static Partition in Hive**

Insert input data files individually into a partition table is Static Partition

Usually when loading files (big files) into Hive tables static partitions are preferred

Static Partition saves your time in loading data compared to dynamic partition

You “statically” add a partition in table and move the file into the partition of the table.

We can alter the partition in static partition

You can get the partition column value form the filename, day of date etc without reading the whole big file.

If you want to use Static partition in hive you should set property set hive.mapred.mode = strict This property set by default in hive-site.xml

Static partition is in Strict Mode

You should use where clause to use limit in static partition

You can perform Static partition on Hive Manage table or external table.

**Dynamic Partition in Hive**

Single insert to partition table is known as dynamic partition

Usually dynamic partition load the data from non partitioned table

Dynamic Partition takes more time in loading data compared to static partition

When you have large data stored in a table then Dynamic partition is suitable.

If you want to partition number of column but you don’t know how many columns then also dynamic partition is suitable

Dynamic partition there is no required where clause to use limit.

we can’t perform alter on Dynamic partition

You can perform dynamic partition on hive external table and managed table

If you want to use Dynamic partition in hive then mode is in nonstrict mode

Here is hive dynamic partition properties you should allow